GPU Computing Lab

LAB-1

Programs-> Hello world, a Kernel Call and Passing Parameters

NAME: **ROMEO SARKAR**
ADMISSION NO.: **20JE0814**
DATE: **10-08-2022**

Experiment 2.1: Programs-> Hello world, a Kernel Call and Passing Parameters.

Objectives: Display “Hello world” on terminal from CPU & GPU through a CUDA Sample Program.

Sample Program:

```
#include <stdio.h>
__global__ void helloFromGPU ()
{
    printf ("Hello World from GPU!\n");
    return;
}
int main ()
{
    printf ("Hello World from CPU!\n");
    helloFromGPU <<<1, 5>>> ();
    cudaDeviceReset ();
    return 0;
}
```

Output:

```
Hello World from CPU!
Hello World from GPU!
Hello World from GPU!
Hello World from GPU!
Hello World from GPU!
Hello World from GPU!
```

Lab Exercise 2.1: Display information from the CPU and GPU as per the followings:

1) Write a CUDA C program to display your 10-10 times, name from CPU and GPU respectively.
2) Write a CUDA C program to display your 4 times Course Name, Name of Experiment and Date from CPU and GPU respectively.

Program_1:

```
#include <stdio.h>
__global__ void print_name ()
{
    for (int i = 0; i < 10; i++)
        printf ("GPU> Romeo Sarkar (%d)\n", i + 1);
    return;
}
int main ()
{
    for (int i = 0; i < 10; i++)
        printf ("CPU> Romeo Sarkar (%d)\n", i + 1);
    print_name <<<1, 1>>> ();
    return 0;
}
```

Output:

```
CPU> Romeo Sarkar (1)
CPU> Romeo Sarkar (2)
CPU> Romeo Sarkar (3)
CPU> Romeo Sarkar (4)
CPU> Romeo Sarkar (5)
CPU> Romeo Sarkar (6)
CPU> Romeo Sarkar (7)
CPU> Romeo Sarkar (8)
CPU> Romeo Sarkar (9)
CPU> Romeo Sarkar (10)
GPU> Romeo Sarkar (1)
GPU> Romeo Sarkar (2)
GPU> Romeo Sarkar (3)
GPU> Romeo Sarkar (4)
GPU> Romeo Sarkar (5)
GPU> Romeo Sarkar (6)
GPU> Romeo Sarkar (7)
GPU> Romeo Sarkar (8)
GPU> Romeo Sarkar (9)
GPU> Romeo Sarkar (10)
```

Program_2:

```
#include <stdio.h>
__global__ void GPU ()
{
    for (int i = 0; i < 4; i++)
        printf ("GPU> (%d)Course Name: GPU Computing Lab; Name of Experiment:
Programs-> Hello world, a Kernel Call and Passing Parameters; Date: %s\n", i +
1, __DATE__);
    return;
}
int main ()
{
    for (int i = 0; i < 4; i++)
        printf ("CPU> (%d)Course Name: GPU Computing Lab; Name of Experiment:
Programs-> Hello world, a Kernel Call and Passing Parameters; Date: %s\n", i +
1, __DATE__);
    GPU <<<1, 1>>> ();
    return 0;
}
```

Output:

```
CPU> (1)Course Name: GPU Computing Lab; Name of Experiment: Programs-> Hello
world, a Kernel Call and Passing Parameters; Date: Aug 14 2022
CPU> (2)Course Name: GPU Computing Lab; Name of Experiment: Programs-> Hello
world, a Kernel Call and Passing Parameters; Date: Aug 14 2022
CPU> (3)Course Name: GPU Computing Lab; Name of Experiment: Programs-> Hello
world, a Kernel Call and Passing Parameters; Date: Aug 14 2022
CPU> (4)Course Name: GPU Computing Lab; Name of Experiment: Programs-> Hello
world, a Kernel Call and Passing Parameters; Date: Aug 14 2022
GPU> (1)Course Name: GPU Computing Lab; Name of Experiment: Programs-> Hello
world, a Kernel Call and Passing Parameters; Date: Aug 14 2022
GPU> (2)Course Name: GPU Computing Lab; Name of Experiment: Programs-> Hello
world, a Kernel Call and Passing Parameters; Date: Aug 14 2022
GPU> (3)Course Name: GPU Computing Lab; Name of Experiment: Programs-> Hello
world, a Kernel Call and Passing Parameters; Date: Aug 14 2022
GPU> (4)Course Name: GPU Computing Lab; Name of Experiment: Programs-> Hello
world, a Kernel Call and Passing Parameters; Date: Aug 14 2022
```

Experiment 2.2: Check Device Information

Objectives: Display information of the first CUDA device including driver version, runtime version, compute capability, bytes of global memory.

Sample Program:

```
#include <cuda_runtime.h>
#include <stdio.h>
int main (int argc, char **argv)
{
    printf ("%s Starting... \n", argv[0]);
    int deviceCount = 0;
    cudaGetDeviceCount (&deviceCount);
    if (deviceCount == 0)
    {
        printf ("There are no available device(s) that support CUDA\n");
    }
    else
    {
        printf ("Detected %d CUDA Capable device(s)\n", deviceCount);
    }
    int dev = 0, driverVersion = 0, runtimeVersion = 0;
    cudaSetDevice (dev);
    cudaDeviceProp deviceProp;
    cudaGetDeviceProperties (&deviceProp, dev);
    printf ("Device: %d: \"%s\"\n", dev, deviceProp.name);
    cudaDriverGetVersion (&driverVersion);
    cudaRuntimeGetVersion (&runtimeVersion);
    printf ("  CUDA Drivers Version / Runtime Version %d.%d / %d.%d\n",
driverVersion / 1000, driverVersion % 100 / 10, runtimeVersion / 1000,
runtimeVersion % 100 / 10);
    printf ("  CUDA Capability Major/Minor version number: %d.%d\n",
deviceProp.major, deviceProp.minor);
    printf ("  Total amount of global memory: %.2f GBytes (%llu " "bytes)\n",
(float) (deviceProp.totalGlobalMem / pow (1024.0, 3)), (unsigned long long)
(deviceProp.totalGlobalMem));
    printf ("  GPU Clock rate: %.0f MHz (%0.2f " "Ghz)\n",
deviceProp.clockRate * 1e-3f, deviceProp.clockRate * 1e-6f);
    printf ("  Memory Clock rate: %.0f Mhz\n", deviceProp.memoryClockRate *
1e-3f);
```

```
    printf ("  Memory Bus Width: %d-bit\n", deviceProp.memoryBusWidth);
    if (deviceProp.l2CacheSize)
    {
        printf ("  L2 Cache Size: %d bytes\n", deviceProp.l2CacheSize);
    }
    printf ("  Max Texture Dimension size: (x, y, z)  1D=(%d), 2D=(%d,%d),
3D=(%d,%d,%d)\n", deviceProp.maxTexture1D,
deviceProp.maxTexture2D[0],  deviceProp.maxTexture2D[1],
deviceProp.maxTexture3D[0], deviceProp.maxTexture3D[1],
deviceProp.maxTexture3D[2]);
    printf ("  Max Layered Texture Size (dim) x layers  1D=(%d) x %d, "
"2D=(%d,%d) x %d\n", deviceProp.maxTexture1DLayered[0],
deviceProp.maxTexture1DLayered[1], deviceProp.maxTexture2DLayered[0],
deviceProp.maxTexture2DLayered[1], deviceProp.maxTexture2DLayered[2]);
    printf ("  Total amount of constant memory:  %zu bytes\n",
deviceProp.totalConstMem);
    printf ("  Total amount of shared memory per block:  %zu bytes\n",
deviceProp.sharedMemPerBlock);
    printf ("  Total number of registers available per block: %d\n",
deviceProp.regsPerBlock);
    exit (EXIT_SUCCESS);
}
```

Output:

```
C:\Users\romeo\OneDrive\Desktop\Windows\My_DATA\COLLEGE_FILES\GPU Computing
Lab\1\sample2.exe Starting...
Detected 1 CUDA Capable device(s)
Device: 0: "NVIDIA GeForce RTX 3060 Laptop GPU"
  CUDA Drivers Version / Runtime Version 11.7 / 11.7
  CUDA Capability Major/Minor version number: 8.6
  Total amount of global memory: 6.00 GBytes (6441926656 bytes)
  GPU Clock rate: 1425 MHz (1.42 Ghz)
  Memory Clock rate: 7001 Mhz
  Memory Bus Width: 192-bit
  L2 Cache Size: 3145728 bytes
  Max Texture Dimension size: (x, y, z)  1D=(131072), 2D=(131072,65536),
3D=(16384,16384,16384)
  Max Layered Texture Size (dim) x layers  1D=(32768) x 2048, 2D=(32768,32768)
x 2048
  Total amount of constant memory:  65536 bytes
  Total amount of shared memory per block:  49152 bytes
  Total number of registers available per block: 65536
```

Lab Exercise 2.2: Write a CUDA program to display the following device information on the terminal:

1) Wrap size:
2) Maximum number of threads per multiprocessor:
3) Maximum number of threads per block:
4) Maximum sizes of each dimension of a block:
5) Maximum sizes of each dimension of a grid:
6) Maximum memory pitch:

Program:

```
#include <cuda_runtime.h>
#include <stdio.h>
int main ()
{
    int deviceCount = 0;
    cudaGetDeviceCount (&deviceCount);
    if (deviceCount == 0)
    {
        printf ("There are no available device(s) that support CUDA\n");
    }
    else
    {
        printf ("Detected %d CUDA Capable device(s)\n", deviceCount);
    }
    int dev = 0;
    cudaSetDevice (dev);
    cudaDeviceProp deviceProp;
    cudaGetDeviceProperties (&deviceProp, dev);
    printf ("Device: %d: \"%s\"\n", dev, deviceProp.name);
    printf ("  Wrap size:                                    %d\n",
deviceProp.warpSize);
    printf ("  Maximum number of threads per multiprocessor:   %d\n",
deviceProp.maxThreadsPerMultiProcessor);
    printf ("  Maximum number of threads per block:           %d\n",
deviceProp.maxThreadsPerBlock);
    printf ("  Maximum sizes of each dimension of a block:    %d x %d x
%d\n", deviceProp.maxThreadsDim[0], deviceProp.maxThreadsDim[1],
deviceProp.maxThreadsDim[2]);
    printf ("  Maximum sizes of each dimension of grid:       %d x %d x
%d\n", deviceProp.maxGridSize[0], deviceProp.maxGridSize[1],
deviceProp.maxGridSize[2]);
    printf ("  Maximum memory pitch:                         %zd bytes\n",
deviceProp.memPitch);
    return 0;
}
```

Output:

```
Detected 1 CUDA Capable device(s)
Device: 0: "NVIDIA GeForce RTX 3060 Laptop GPU"
  Wrap size:                                    32
  Maximum number of threads per multiprocessor:   1536
  Maximum number of threads per block:           1024
  Maximum sizes of each dimension of a block:    1024 x 1024 x 64
  Maximum sizes of each dimension of grid:       2147483647 x 65535 x 65535
  Maximum memory pitch:                         2147483647 bytes
```

Learning Outcomes: To write program to understand host, device and global functions.

----------X----------